Ingénierie des systèmes d'information. CRDT.

Sergey Kirgizov

ESIREM, LIB

2021

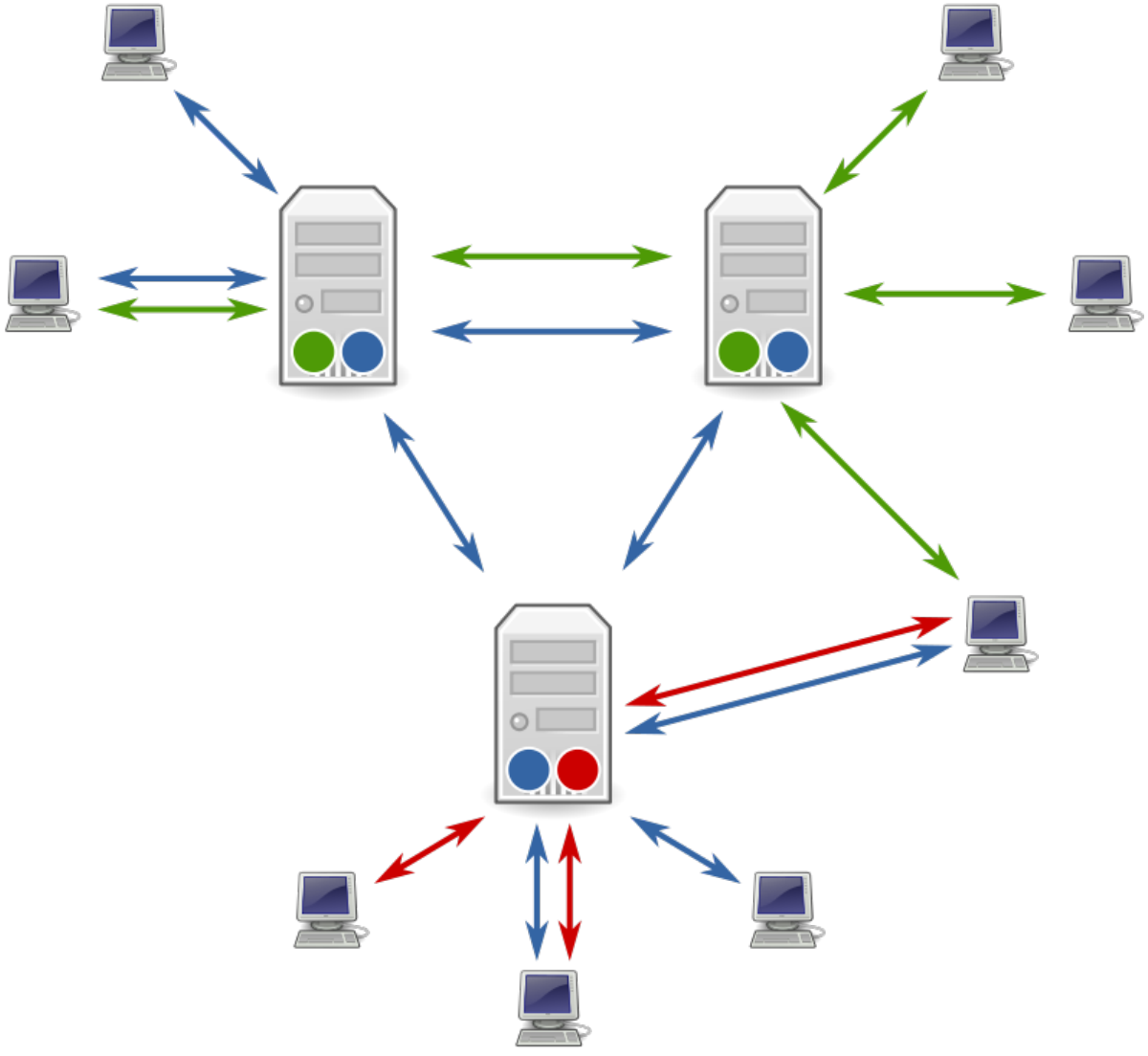
Plan

- Systèmes distribués
- Théorème CAP
- ACID vs BASE
- Cohérence à terme / eventual consistency
- State-based CRDTs
  - G-counter
  - PN-counter
  - G-set
  - 2P-set
  - LWW-Element-Set
  - OR-set
- Exemples

Systèmes distribués

Systèmes distribués. Exemple I

Usenet



src : Benjamin D. Esham @ Wikipedia

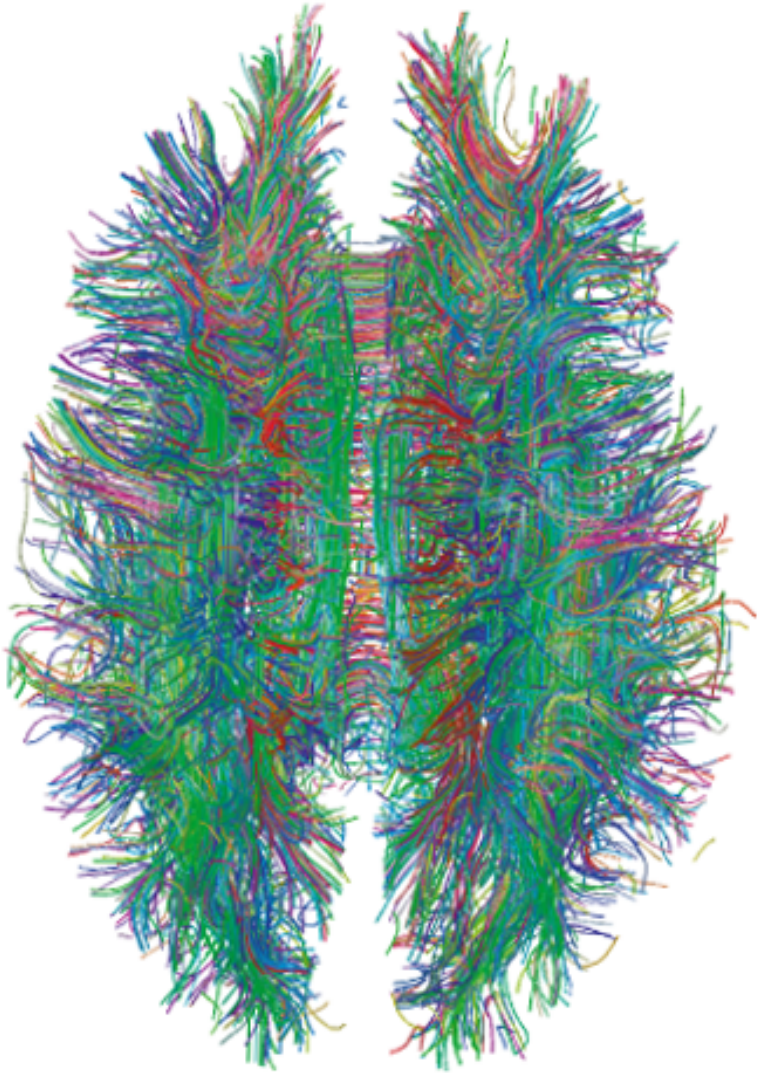
Systèmes distribués. Exemple II

Société

src : chensiyuan @ Wikipedia

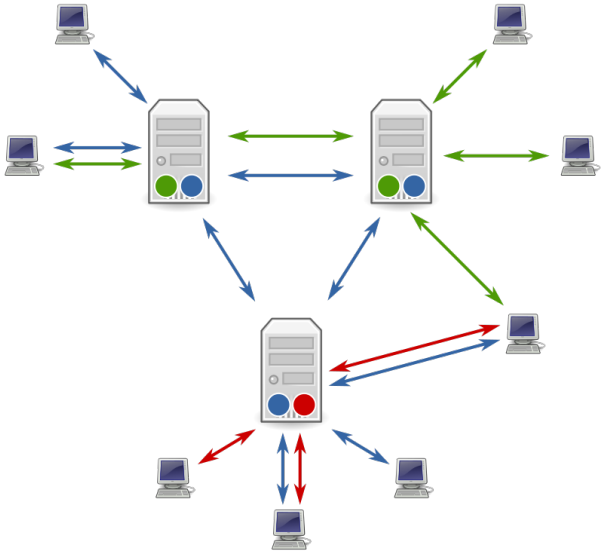
Systèmes distribués. Exemple III

Cerveau



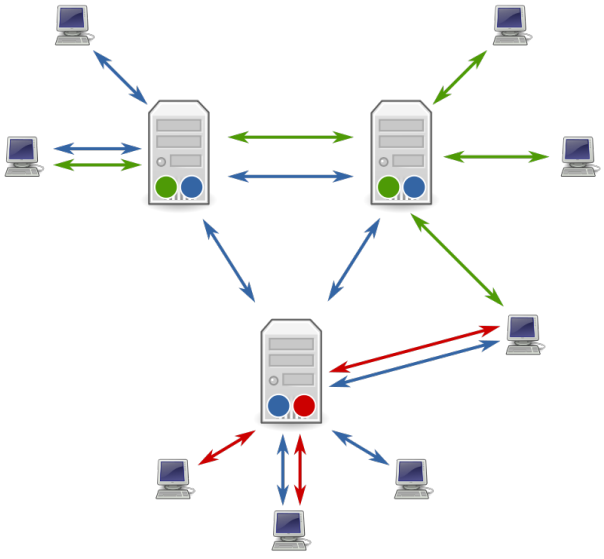
src : Gigandet X, Hagmann P, Kurant M, Cammoun L, Meuli R, et al. Estimating the Confidence Level of White Matter Connections Obtained with MRI Tractography. PLoS ONE 3(12) : e4006.

En informatique



Système distribué

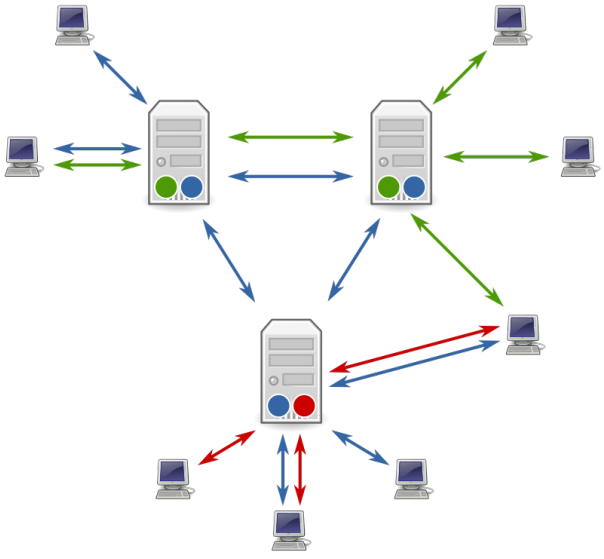
En informatique



Système distribué

- les nœuds du système communiquent via des messages

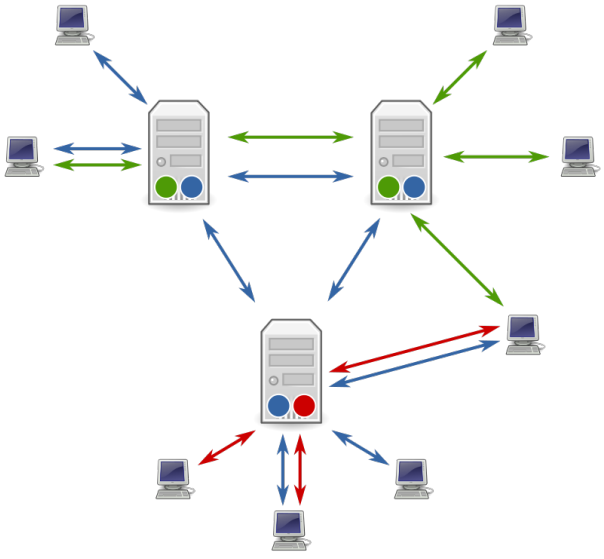
En informatique



Système distribué

- les nœuds du système communiquent via des messages
- l'utilisateur peut demander l'état d'un nœud

En informatique



Système distribué

- les nœuds du système communiquent via des messages
- l'utilisateur peut demander l'état d'un nœud
- l'utilisateur peut mettre à jour l'état d'un nœud

Théorème CAP

Théorème CAP

Au plus deux parmi les propriétés suivantes peuvent être garanties par un système :

- (**C**onsistency) Cohérence
  Tous les nœuds du système voient exactement les mêmes données au même moment
- (**A**vailability) Disponibilité
  Garantie que toutes les requêtes reçoivent une réponse
- (**P**artition Tolerance) Tolérance au partitionnement
  Aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement

src : `https://en.wikipedia.org/wiki/CAP_theorem`

Théorème CAP. Histoire

- Il a été conjecturé par Eric Brewer (Berkeley, Google) en 2000. Slides d'Eric : `https://people.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf`

- Seth Gilbert and Nancy Lynch (MIT) ont donné une preuve de cette conjecturé. `http://groups.csail.mit.edu/tds/papers/Gilbert/Brewer6.pdf`

ACID vs BASE

ACID vs BASE

ACID

- Atomicity
- Consistency
- Isolation
- Durability

ACID vs BASE

ACID

- Atomicity
- Consistency
- Isolation
- Durability

BASE

- Basic Availability
- Soft-state
- Eventual consistency

Les deux approches peuvent être décentralisées.
ACID préfère cohérence.

BASE préfère disponibilité :

ACID vs BASE

ACID

- Atomicity
- Consistency
- Isolation
- Durability

BASE

- Basic Availability
- Soft-state
- Eventual consistency

Les deux approches peuvent être décentralisées.
ACID préfère cohérence.

BASE préfère disponibilité :

Les opérations de lecture et d'écriture sont disponibles autant que possible sur tous les nœuds. Réponses approximatives (parfois non cohérentes à 100%) sont ok. Au fil du temps, les réponses ont tendance à être de plus en plus cohérentes. Finalement, ça doit converger vers des valeurs 100% cohérentes.

Voir aussi : un article de Charles Roe `https://www.dataversity.net/acid-vs-base-the-shifting-ph-of-database-transaction-processing/`

Cohérence à terme / eventual consistency

Cohérence à terme / eventual consistency

Système distribué :

- les nœuds du système communiquent via des messages
- l'utilisateur peut demander l'état d'un nœud
- l'utilisateur peut mettre à jour l'état d'un nœud

Cohérence à terme / eventual consistency

Système distribué :

- les nœuds du système communiquent via des messages
- l'utilisateur peut demander l'état d'un nœud
- l'utilisateur peut mettre à jour l'état d'un nœud

Si un nœud a été mis à jour, les autres nœuds devraient voir cette mise à jour, mais peut-être pas tout de suite.

Cohérence à terme / eventual consistency

Système distribué :

- les nœuds du système communiquent via des messages
- l'utilisateur peut demander l'état d'un nœud
- l'utilisateur peut mettre à jour l'état d'un nœud

Si un nœud a été mis à jour, les autres nœuds devraient voir cette mise à jour, mais peut-être pas tout de suite.

L'ordre de réception des mises à jour peut être différent pour différents nœuds.

Cohérence à terme / eventual consistency

Système distribué :

- les nœuds du système communiquent via des messages
- l'utilisateur peut demander l'état d'un nœud
- l'utilisateur peut mettre à jour l'état d'un nœud

Si un nœud a été mis à jour, les autres nœuds devraient voir cette mise à jour, mais peut-être pas tout de suite.

L'ordre de réception des mises à jour peut être différent pour différents nœuds.

C'est le modèle de cohérence le plus faible utilisé en pratique.

fr.wikipedia.org/wiki/Cohérence_(données)#Cohérence_à_terme

Strong eventual consistency

"any two nodes that have received the same (unordered) set of updates will be in the same state"

en.wikipedia.org/wiki/Eventual_consistency#Strong_eventual_consistency

Strong eventual consistency

"any two nodes that have received the same (unordered) set of updates will be in the same state"

en.wikipedia.org/wiki/Eventual_consistency#Strong_eventual_consistency

Donc, l'opération de fusion des mises à jour devrait être

- commutative,
- associative,
- et, si on peut avoir des répétitions, idempotente.

Conflict-free Replicated Data Types (CRDT)

- **Efficient Solutions to the Replicated Log and Dictionary Problems** Gene T.J. Wuu et Arthur J. Bernstein, 1984
- **Conflict-free Replicated Data Types** Marc Shapiro, Nuno Preguiça, Carlos Baquero et Marek Zawirski, 2011

Quelques articles sur le sujet :

`https://kirgizov.link/teaching/esirem/information-systems-2021/CRDT/`

Page de Marc Shapiro

`https://pages.lip6.fr/Marc.Shapiro/`

CRDT

- **State-based CRDT (convergent replicated data type, CvRDT)**
  Transmission des états complets entre les nœuds.
  L'infrastructure de communication garantir que tous les messages sont livrés au mois une fois, mais ils peuvent être livrés dans n'importe quel ordre, et il peut y avoir des doublons.

CRDT

- **State-based CRDT (convergent replicated data type, CvRDT)**
  Transmission des états complets entre les nœuds.
  L'infrastructure de communication garantir que tous les messages sont livrés au mois une fois, mais ils peuvent être livrés dans n'importe quel ordre, et il peut y avoir des doublons.

  Exemples : systèmes de fichiers NFS, AFS et Coda ; bases de données Dynamo d'Amazon, Riak, Roshi de SoundCloud, Cosmos DB de Microsoft, etc...

CRDT

- **State-based CRDT (convergent replicated data type, CvRDT)**
  Transmission des états complets entre les nœuds.
  L'infrastructure de communication garantir que tous les messages sont livrés au mois une fois, mais ils peuvent être livrés dans n'importe quel ordre, et il peut y avoir des doublons.

  Exemples : systèmes de fichiers NFS, AFS et Coda ; bases de données Dynamo d'Amazon, Riak, Roshi de SoundCloud, Cosmos DB de Microsoft, etc...

- **Operation-based CRDT (commutative replicated data Type, CmRDT)**
  Transmission des mises à jour entre les nœuds.
  L'infrastructure de communication garantit que tous les messages sont livrés une seule fois, sans duplication, mais ils peuvent être dans n'importe quel ordre.

  Exemples : systèmes coopératifs : Bayou, Rover, IceCube, Telex, etc...

State-based CRDTs

State-based CRDTs

- G-counter
- PN-counter
- G-set
- 2P-set
- LWW-Element-Set
- OR-Set
- Arbres, graphes, etc...

Documentation :

- `https://en.wikipedia.org/wiki/Conflict-free_replicated_data_type`
- Un article **"A comprehensive study of Convergent and Commutative Replicated Data Types"**
  de Marc Shapiro, Nuno Preguiça, Carlos Baquero et Marek Zawirski
  `https://kirgizov.link/teaching/esirem/information-systems-2021/CRDT/CRDT2.pdf`

Grow only Counter (G-counter)

Specification 6 State-based increment-only counter (vector version)

```
1:  payload integer[n] P                                  ▷ One entry per replica
2:      initial [0, 0, ..., 0]
3:  update increment ()
4:      let g = myID()                                    ▷ g: source replica
5:      P[g] := P[g] + 1
6:  query value () : integer v
7:      let v = Σ_i P[i]
8:  compare (X, Y) : boolean b
9:      let b = (∀i ∈ [0, n − 1] : X.P[i] ≤ Y.P[i])
10: merge (X, Y) : payload Z
11:     let ∀i ∈ [0, n − 1] : Z.P[i] = max(X.P[i], Y.P[i])
```

max est commutative, associative, et idempotente.

Positive-negative counter (PN-counter)

Comment faire un compteur qui peut croître et décroître ?

Positive-negative counter (PN-counter)

Comment faire un compteur qui peut croître et décroître ?
Combiner deux G-counters !

Positive-negative counter (PN-counter)

Comment faire un compteur qui peut croître et décroître ?
Combiner deux G-counters !

Specification 7 State-based PN-Counter

```
1:  payload integer[n] P, integer[n] N                          ▷ One entry per replica
2:      initial [0, 0, ..., 0], [0, 0, ..., 0]
3:  update increment ()
4:      let g = myID()                                          ▷ g: source replica
5:      P[g] := P[g] + 1
6:  update decrement ()
7:      let g = myID()
8:      N[g] := N[g] + 1
9:  query value () : integer v
10:     let v = Σ_i P[i] − Σ_i N[i]
11: compare (X, Y) : boolean b
12:     let b = (∀i ∈ [0, n − 1] : X.P[i] ≤ Y.P[i] ∧ ∀i ∈ [0, n − 1] : X.N[i] ≤ Y.N[i])
13: merge (X, Y) : payload Z
14:     let ∀i ∈ [0, n − 1] : Z.P[i] = max(X.P[i], Y.P[i])
15:     let ∀i ∈ [0, n − 1] : Z.N[i] = max(X.N[i], Y.N[i])
```

Grow only set (G-set)

Specification 11 State-based grow-only Set (G-Set)

```
 1: payload set A
 2:     initial ∅
 3: update add (element e)
 4:     A := A ∪ {e}
 5: query lookup (element e) : boolean b
 6:     let b = (e ∈ A)
 7: compare (S, T) : boolean b
 8:     let b = (S.A ⊆ T.A)
 9: merge (S, T) : payload U
10:     let U.A = S.A ∪ T.A
```

$\cup$ est commutative, associative, et idempotente.

Two-Phase set (2P-set)

Comment créer une structure dans laquelle on peut non seulement ajouter mais aussi supprimer des éléments ?

Two-Phase set (2P-set)

Comment créer une structure dans laquelle on peut non seulement ajouter mais aussi supprimer des éléments ?
Combiner deux G-sets !

Two-Phase set (2P-set)

Comment créer une structure dans laquelle on peut non seulement ajouter mais aussi supprimer des éléments ?
Combiner deux G-sets !

Specification 12 State-based 2P-Set

```
1:  payload set A, set R                                  ▷ A: added; R: removed
2:      initial ∅, ∅
3:  query lookup (element e) : boolean b
4:      let b = (e ∈ A ∧ e ∉ R)
5:  update add (element e)
6:      A := A ∪ {e}
7:  update remove (element e)
8:      pre lookup(e)
9:      R := R ∪ {e}
10: compare (S, T) : boolean b
11:     let b = (S.A ⊆ T.A ∨ S.R ⊆ T.R)
12: merge (S, T) : payload U
13:     let U.A = S.A ∪ T.A
14:     let U.R = S.R ∪ T.R
```

Two-Phase set (2P-set)

Comment créer une structure dans laquelle on peut non seulement ajouter mais aussi supprimer des éléments ?
Combiner deux G-sets !

Specification 12 State-based 2P-Set

payload set A, set R ▷ A: added; R: removed
    initial $\varnothing, \varnothing$
query *lookup* (element e) : boolean b
    let $b = (e \in A \wedge e \notin R)$
update *add* (element e)
    $A := A \cup \{e\}$
update *remove* (element e)
    pre *lookup*(e)
    $R := R \cup \{e\}$
compare (S, T) : boolean b
    let $b = (S.A \subseteq T.A \vee S.R \subseteq T.R)$
merge (S, T) : payload U
    let $U.A = S.A \cup T.A$
    let $U.R = S.R \cup T.R$

Problème : une fois que l'élément est supprimé, on ne peut pas le rajouter !

Last-Write-Wins-Element-Set (LWW-Element-Set)

C'est comme 2P-set mais nous associons à chaque élément des étiquettes-timestamps.

L'ordre des messages doit respecter la causalité !

Observed-Remove Set (OR-set)

Pour chaque élément de l'ensemble, un ensemble des étiquettes uniques est maintenu ainsi que un ensemble des étiquettes liées aux éléments supprimés.

Observed-Remove Set (OR-set)

Pour chaque élément de l'ensemble, un ensemble des étiquettes uniques est maintenu ainsi que un ensemble des étiquettes liées aux éléments supprimés.

Problème : si le même élément est ajouté et supprimé plusieurs fois, la taille va augmenter de manière illimitée !

Observed-Remove Set (OR-set)

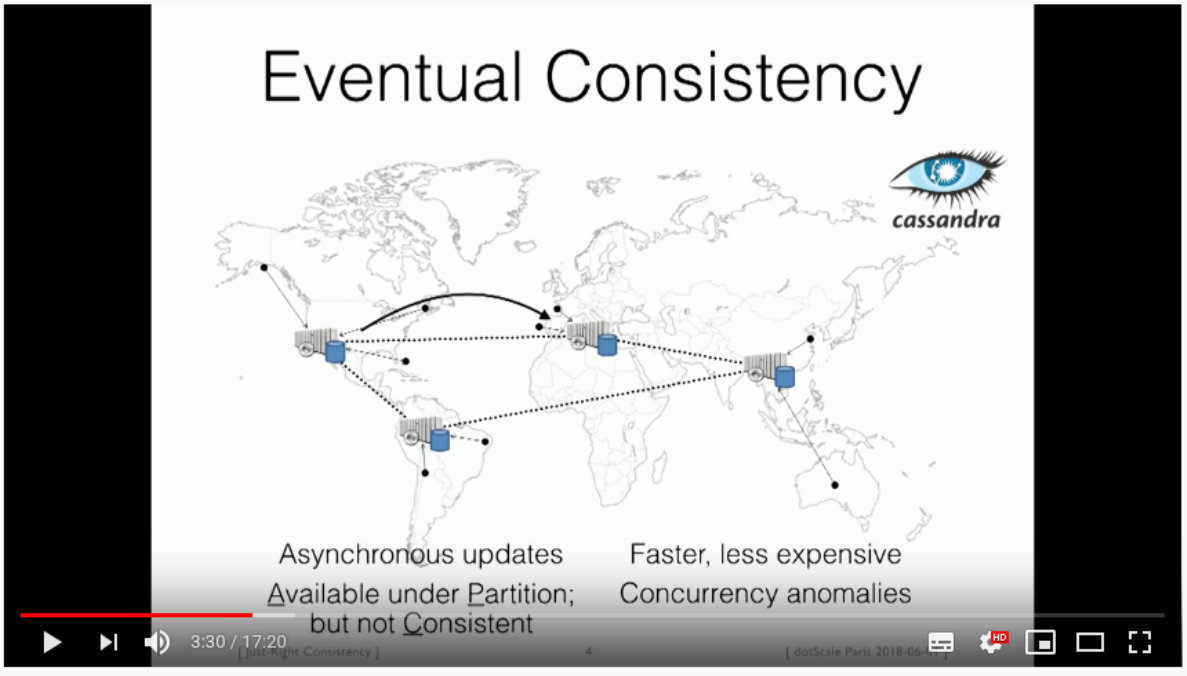
Pour chaque élément de l'ensemble, un ensemble des étiquettes uniques est maintenu ainsi que un ensemble des étiquettes liées aux éléments supprimés.

Problème : si le même élément est ajouté et supprimé plusieurs fois, la taille va augmenter de manière illimitée !

L'optimisation existe.
Annette Bieniusa, Marek Zawirski, Nuno Preguiça, Marc Shapiro, Carlos Baquero, Valter Balegas, Sérgio Duarte
"An Optimized Conflict-free Replicated Set"
`https://arxiv.org/abs/1210.3368`

S’il y a des ensembles,
il y a tout le reste.

Un talk de Marc Shapiro



dotScale 2018 - Marc Shapiro - Just-Right Consistency

`https://www.youtube.com/watch?v=lO-UfHASUSE`

Exemples

Riak

riak

C'est un SGBD distribué orienté clé-valeur.
Programmé en Erlang.
Erlang est utilisé pour programmer des systèmes distribués.

Industry use [edit]

Nimbus Note is a collaborative note-taking application that uses the Yjs CRDT for collaborative editing. [23]

Redis is a distributed, highly available and scalable in-memory database that uses CRDTs for implementing globally distributed databases based on and fully compatible with Redis open source.

SoundCloud open-sourced Roshi, a LWW-element-set CRDT for the SoundCloud stream implemented on top of Redis.[24]

Riak is a distributed NoSQL key-value data store based on CRDTs.[25] League of Legends uses the Riak CRDT implementation for its in-game chat system, which handles 7.5 million concurrent users and 11,000 messages per second.[26] Bet365, stores hundreds of megabytes of data in the Riak implementation of OR-Set.[27]

TomTom employs CRDTs to synchronize navigation data between the devices of a user.[28]

Phoenix, a web framework written in Elixir, uses CRDTs to support real time multi-node information sharing in version 1.2.[29]

Facebook implements CRDTs in their Apollo low-latency "consistency at scale" database.[30]

Teletype for Atom employs CRDTs to enable developers share their workspace with team members and collaborate on code in real time.[31]

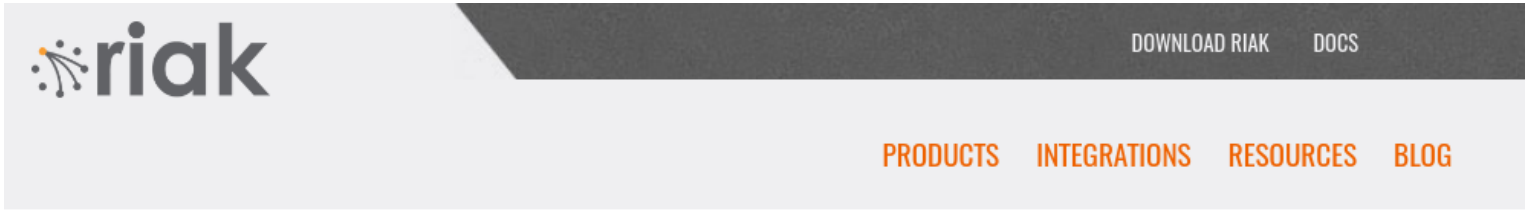
Haja Networks' OrbitDB uses operation-based CRDTs in its core data structure, IPFS-Log.[32]

Apple implements CRDTs in the Notes app for syncing offline edits between multiple devices.[33]

https://en.wikipedia.org/wiki/Conflict-free_replicated_data_type#Industry_use

Avec CRDT !

`https://riak.com/products/riak-kv/riak-distributed-data-types/`



Simplify development of distributed applications

Riak KV is a distributed system architected to never lose a write, so conflicts between replicas are inevitable. Riak Distributed Data Types reduce the complexity of building distributed applications by providing built-in conflict resolution in the Riak Data Types themselves.

While Riak KV is built as a data-agnostic key/value store, Riak Data Types enable you to use Riak KV as a data-aware system and perform transactions on CRDT-inspired data types. The following Riak Distributed Data Types are implemented in Riak KV:

- Flags
- Registers
- Counters
- Sets
- Maps
- HyperLogLog

Merci.